**Very-Long Instruction Word (VLIW) Computer Architecture**

*An Introduction To Very-Long Instruction Word (VLIW) Computer Architecture by Philips Semiconductors*

**ABSTRACT**

VLIW implementations are simpler for very high performance.

Việc triển khai VLIW đơn giản hơn cho hiệu suất rất cao.

Just as RISC architectures permit simpler, cheaper high-performance implementations than do CISCs, VLIW architectures are simpler and cheaper than RISCs because of further hardware simplifications.

Giống như các kiến ​​trúc RISC cho phép triển khai hiệu suất cao đơn giản hơn, rẻ hơn so với các CISC, các kiến ​​trúc VLIW đơn giản và rẻ hơn so với RISC bởi vì đơn giản hóa phần cứng hơn nữa.

VLIW architectures, however, require more compiler support.

Kiến trúc VLIW, tuy nhiên, yêu cầu hỗ trợ trình biên dịch nhiều hơn.

**INTRODUCTION AND MOTIVATION**

**WHY VLIW?**

The key to higher performance in microprocessors for a broad range of applications is the ability to exploit fine-grain, instruction-level parallelism. Some methods for exploiting fine-grain parallelism include:

+ pipelining

+ multiple processors

+ superscalar implementation

+ specifying multiple independent operations per instruction

Chìa khóa để có hiệu suất cao hơn trong các bộ vi xử lý cho một phạm vi rộng các ứng dụng là khả năng khai thác song song mức độ hạt mịn. Một số phương pháp khai thác song song hạt mịn bao gồm:

+ đường ống

+ nhiều bộ xử lý

+ thực hiện siêu khối

+ chỉ định nhiều hoạt động độc lập cho mỗi hướng dẫn

Pipelining is now universally implemented in high-performance processors. Little more can be gained by improving the implementation of a single pipeline.

Pipelining hiện được thực hiện phổ biến trong các bộ xử lý hiệu suất cao. Ít nhiều có thể đạt được bằng cách cải thiện việc thực hiện một đường ống duy nhất.

Using multiple processors improves performance for only a restricted set of applications.

Sử dụng nhiều bộ xử lý sẽ cải thiện hiệu suất chỉ cho một bộ ứng dụng bị hạn chế.

Superscalar implementations can improve performance for all types of applications. Superscalar (super: beyond; scalar: one dimensional) means the ability to fetch, issue to execution units, and complete more than one instruction at a time. Superscalar implementations are required when architectural compatibility must be preserved, and they will be used for entrenched architectures with legacy software, such as the x86 architecture that dominates the desktop computer market.

Việc triển khai Superscalar có thể cải thiện hiệu suất cho tất cả các loại ứng dụng. Superscalar (super: beyond; vô hướng: một chiều) có nghĩa là khả năng tìm nạp, phát hành cho các đơn vị thực thi và hoàn thành nhiều hơn một lệnh cùng một lúc. Việc triển khai siêu khối được yêu cầu khi khả năng tương thích kiến trúc phải được bảo tồn và chúng sẽ được sử dụng cho các kiến trúc cố thủ với phần mềm cũ, chẳng hạn như x86 kiến trúc thống trị thị trường máy tính để bàn.

Specifying multiple operations per instruction creates a very-long instruction word architecture or VLIW. A VLIW implementation has capabilities very similar to those of a superscalar processor—issuing and completing more than one operation at a time—with one important exception: the VLIW hardware is not responsible for discovering opportunities to execute multiple operations concurrently. For the VLIW

implementation, the long instruction word already encodes the concurrent operations. This explicit encoding leads to dramatically reduced hardware complexity compared to a high-degree superscalar implementation of a RISC or CISC.

Chỉ định nhiều thao tác cho mỗi lệnh tạo ra kiến trúc từ lệnh rất dài hoặc VLIW. Việc triển khai VLIW có các khả năng rất giống với khả năng của bộ xử lý siêu tốc độ phát hành và hoàn thành nhiều hơn một thao tác tại một thời điểm với một ngoại lệ quan trọng: phần cứng VLIW không chịu trách nhiệm phát hiện nhiều cơ hội để thực hiện đồng thời nhiều hoạt động. Đối với VLIW thực hiện, từ lệnh dài đã mã hóa các hoạt động đồng thời. Mã hóa rõ ràng này dẫn đến giảm đáng kể độ phức tạp phần cứng so với việc triển khai siêu cấp độ cao của RISC hoặc CISC.

* The big advantage of VLIW, then, is that a highly concurrent (parallel) implementation is much simpler and cheaper to build than equivalently concurrent RISC or CISC chips. VLIW is a simpler way to build a superscalar microprocessor.
* Do đó, lợi thế lớn của VLIW là việc triển khai đồng thời (song song) rất đơn giản và rẻ hơn để xây dựng so với các chip RISC hoặc CISC đồng thời tương đương. VLIW là một cách đơn giản hơn để xây dựng bộ vi xử lý siêu khối.

**ARCHITECTURE VS. IMPLEMENTATION**

The word architecture in the context of computer science is often misused. Used accurately, architecture refers to the instruction set and resources available to someone who writes programs. The architecture is what is described in a definition document, often called a user’s manual. Thus, architecture contains instruction formats, instruction semantics (operation definitions), registers, memory addressing modes, characteristics of the address space (linear, segmented, special address regions), and anything else a programmer would need to know.

Kiến trúc từ trong bối cảnh khoa học máy tính thường bị sử dụng sai. Được sử dụng chính xác, kiến trúc đề cập đến tập lệnh và tài nguyên có sẵn cho người viết chương trình. Kiến trúc là những gì được mô tả trong một tài liệu định nghĩa, thường được gọi là hướng dẫn sử dụng. Do đó, kiến trúc chứa các định dạng lệnh, ngữ nghĩa lệnh (định nghĩa hoạt động), thanh ghi, chế độ địa chỉ bộ nhớ, đặc điểm của không gian địa chỉ (tuyến tính, phân đoạn, vùng địa chỉ đặc biệt) và bất cứ thứ gì khác a lập trình viên sẽ cần phải biết.

An implementation is the hardware design that realizes the operations specified by the architecture. The implementation determines the characteristics of a microprocessor that are most often measured: price, performance, power consumption, heat dissipation, numbers of pins, operating frequency, and so on.

Một triển khai là thiết kế phần cứng thực hiện các hoạt động được chỉ định bởi kiến trúc. Việc thực hiện xác định các đặc tính của bộ vi xử lý thường được đo lường nhất: giá cả, hiệu suất, mức tiêu thụ điện năng, tản nhiệt, số lượng chân, tần số hoạt động, v.v.

Architecture and implementation are separate, but they do interact. As many researchers into computer architecture discovered between the mid 1970s and 1980s, architecture can have a dramatic effect on the quality of an implementation. In the mid 1980s, IC process technology could fabricate a microcoded implementation of a CISC instruction set and a tiny cache or MMU. For about the same cost, this same process technology could fabricate a pipelined implementation of a simple RISC instruction set (including large register file) with an MMU. At the time, however, chip technology was not dense enough to build a cost-effective, pipelined implementation of a CISC instruction set. As a result, pipelined RISC chips enjoyed a dramatic performance advantage and made a compelling case for RISC architectures.

Kiến trúc và thực hiện là riêng biệt, nhưng chúng tương tác. Như nhiều nhà nghiên cứu về kiến trúc máy tính được phát hiện từ giữa những năm 1970 đến 1980, kiến trúc có thể có tác động mạnh mẽ đến chất lượng của một triển khai. Vào giữa những năm 1980, công nghệ xử lý vi mạch có thể chế tạo một triển khai vi mã hóa của tập lệnh CISC và bộ đệm hoặc MMU nhỏ. Với cùng một chi phí, công nghệ xử lý tương tự này có thể chế tạo việc triển khai theo đường ống của một tập lệnh RISC đơn giản (bao gồm tệp đăng ký lớn) với MMU. Tuy nhiên, tại thời điểm đó, công nghệ chip không đủ dày đặc để xây dựng một triển khai thực hiện theo đường ống, hiệu quả về chi phí của một tập lệnh CISC. Do đó, các chip RISC có đường ống rất thích một lợi thế hiệu suất đáng kinh ngạc và tạo ra một trường hợp hấp dẫn cho các kiến trúc RISC.

The important point is that a range of implementations of any architecture can be built, but architecture influences the quality and cost-effectiveness of those implementations. The influence is exerted largely in the trade-offs that must be made to accommodate complexity associated with the instruction set. The more chip area spent on logic to decode instructions and implement irregularities, the less that can be devoted to performance-enhancing features.

Điểm quan trọng là một loạt các triển khai của bất kỳ kiến trúc nào có thể được xây dựng, nhưng kiến trúc ảnh hưởng đến chất lượng và hiệu quả chi phí của các triển khai đó. Ảnh hưởng được tác động phần lớn trong sự đánh đổi phải được thực hiện để phù hợp với sự phức tạp liên quan đến tập lệnh. Khu vực chip dành nhiều logic hơn để giải mã các hướng dẫn và thực hiện các bất thường, càng ít có thể dành cho tính năng nâng cao hiệu suất.

**ARCHITECTURE COMPARISON: CISC, RISC, AND VLIW**

From the larger perspective, RISC, CISC, and VLIW architectures have more similarities than differences. The differences that exist, however, have profound effects on the implementations of these architectures.

Từ góc nhìn lớn hơn, các kiến trúc RISC, CISC và VLIW có nhiều điểm tương đồng hơn là khác biệt. Tuy nhiên, sự khác biệt tồn tại có ảnh hưởng sâu sắc đến việc triển khai các kiến trúc này.

Obviously these architectures all use the traditional state-machine model of computation: Each instruction effects an incremental change in the state (memory, registers) of the computer, and the hardware fetches and executes instructions sequentially until a branch instruction causes the flow of control to change.

Rõ ràng tất cả các kiến trúc này đều sử dụng mô hình tính toán máy trạng thái truyền thống: Mỗi lệnh thực hiện một thay đổi gia tăng về trạng thái (bộ nhớ, thanh ghi) của máy tính, và phần cứng tìm nạp và thực hiện các lệnh theo tuần tự cho đến khi một lệnh điều khiển nhánh gây ra luồng điều khiển thay đổi.

|  |  |  |  |
| --- | --- | --- | --- |
| ARCHITECTURE  CHARACTERISTIC | CISC | RISC | VLIW |
| INSTRUCTION SIZE | Varies | One size, usually 32 bits | One size |
| INSTRUCTION FORMAT | Field placement varies | Regular, consistent  placement of fields | Regular, consistent placement of  fields |
| INSTRUCTION  SEMANTICS  HƯỚNG DẪN SỬ DỤNG | Varies from simple to  complex; possibly many dependent operations  per instruction  Khác nhau từ đơn giản đến phức tạp; có thể nhiều hoạt động phụ thuộc trên mỗi hướng dẫn | Almost always one simple operation | Many simple, independent  operations |
| REGISTERS | Few, sometimes special | Many, general-purpose | Many, general-purpose |
| MEMORY REFERENCES | Bundled with operations in many different types of  instructions | Not bundled with  operations, i.e., load/store architecture | Not bundled with operations, i.e., load/store architecture |
| HARDWARE DESIGN  FOCUS | Exploit microcoded  Implementations  Khai thác triển khai mã hóa | Exploit implementations  with one pipeline and & no microcode  Khai thác triển khai với một đường ống và & không có vi mã | Exploit implementations with  multiple pipelines, no microcode & no complex dispatch logic  Khai thác triển khai với nhiều đường ống, không có vi mã & không có logic gửi phức tạp |
| C:\Users\thaihuuvuong\Desktop\draftasdasd.PNG | | | |

Table 1.

**The differences between RISC, CISC, and VLIW are in the formats and semantics of the instructions. Table 1compares architecture characteristics.**

**Sự khác biệt giữa RISC, CISC và VLIW nằm ở formats và semantics of the instructions. Bảng 1 đặc điểm kiến trúc.**

**CISC instructions vary in size, often specify a sequence of operations, and can require serial (slow) decoding algorithms. CISCs tend to have few registers, and the registers may be special-purpose, which restricts the ways in which they can be used. Memory references are typically combined with other operations (such as add memory to register). CISC instruction sets are designed to take advantage of microcode.**

**Các CISC instructions khác nhau về kích thước, thường chỉ định một chuỗi các hoạt động và có thể yêu cầu các thuật toán giải mã nối tiếp (chậm). CISC có ít registers, và các registers có thể có mục đích đặc biệt, điều này hạn chế các cách sử dụng chúng. Tham chiếu bộ nhớ thường được kết hợp với các hoạt động khác (chẳng hạn như thêm bộ nhớ vào register). Các bộ hướng dẫn CISC được thiết kế để tận dụng lợi thế của microcode.**

**RISC instructions specify simple operations, are fixed in size, and are easy (quick) to decode. RISC architectures have a relatively large number of general-purpose registers. Instructions can reference main memory only through simple load-register-from-memory and store-register-to-memory operations. RISC instruction sets do not need microcode and are designed to simplify pipelining.**

**Các RISC instructions chỉ định các thao tác đơn giản, được cố định về kích thước và dễ dàng (nhanh chóng) để giải mã. Kiến trúc RISC có số lượng general-purpose registers tương đối lớn. Các Instructions chỉ có thể tham chiếu bộ nhớ chính thông qua các hoạt động đơn giản load-register-from-memory và store-register-to-memory. Các tập lệnh của RISC không cần microcode và được thiết kế để đơn giản hóa pipelining.**

**VLIW instructions are like RISC instructions except that they are longer to allow them to specify multiple, independent simple operations. A VLIW instruction can be thought of as several RISC instructions joined together. VLIW architectures tend to be RISC-like in most attributes.**

**Các VLIW instructions giống như các RISC instructions ngoại trừ việc chúng dài hơn để cho phép chúng chỉ định nhiều hoạt động đơn giản, độc lập. Một VLIW instruction có thể được coi là một số RISC instructions được nối với nhau. Các kiến trúc VLIW có xu hướng giống RISC trong hầu hết các thuộc tính.**

**Figure 1 shows a C-language code fragment containing small function definition. This function adds a local variable to a parameter passed from the caller of the function.**

**Hình 1 cho thấy một đoạn mã ngôn ngữ C chứa định nghĩa hàm nhỏ. Hàm này cộng một biến cục bộ vào một tham số được truyền từ lời gọi hàm.**

**The implementation of this function in CISC, RISC, and VLIW code is also shown. This example is extremely unfair to the RISC and VLIW machines, but it illustrates the differences between the architectures.**

**Triển khai chức năng thông qua các đoạn mã CISC, RISC và VLIW. Ví dụ này cực kỳ không công bằng đối với các máy RISC và VLIW, nhưng nó minh họa sự khác biệt giữa các kiến trúc.**

**The CISC code consists of one instruction because the CISC architecture has an add instruction that can encode a memory address for the destination. So, the CISC instruction adds the local variable in register r2 to the memory-based parameter. The encoding of this CISC instruction might take four bytes on some hypothetical machine.**

**Mã CISC bao gồm một instruction vì kiến trúc CISC có ADD instruction có thể mã hóa địa chỉ bộ nhớ cho đích. Vì vậy, CISC instruction ADD biến cục bộ trong thanh ghi r2 vào tham số dựa trên bộ nhớ. Mã hóa của hướng dẫn CISC này có thể mất bốn byte trên một số máy giả định.**

**The RISC code is artificially inefficient. Normally, a good compiler would pass the parameter in a register, which would make the RISC code consist of only a single register-to-register add instruction. For the sake of illustration, however, the code will consist of three instructions as shown. These three instructions load the parameter to a register, add it to the local variable already in a register, and then store the result back to memory. Each RISC instruction requires four bytes.**

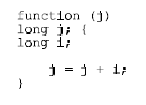
**Mã RISC không hiệu quả. Thông thường, một trình biên dịch tốt sẽ truyền tham số trong một register, điều này sẽ làm cho mã RISC chỉ bao gồm một lệnh ADD register-to-register. Tuy nhiên, để minh họa, mã sẽ bao gồm ba instructions như được hiển thị. Ba hướng dẫn này tải tham số vào một thanh ghi, ADD nó vào biến cục bộ đã có trong một thanh ghi, sau đó lưu kết quả trở lại bộ nhớ. Mỗi lệnh RISC yêu cầu bốn byte.**

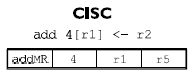
**The VLIW code is similarly hampered by poor register allocation. The example VLIW architecture shown has the ability to simultaneously issue three operations. The first slot (group of four bytes) is for branch instructions, the middle slot is for ALU instructions, and the last slot is for the load/store unit. Since the three RISC operations needed to implement the code fragment are dependent, it is not possible to pack the load and add in the same VLIW instruction. Thus, three separate VLIW instructions are necessary.**

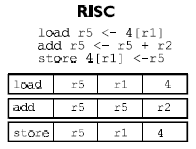
**Mã VLIW bị cản trở tương tự do phân bổ đăng ký kém. Kiến trúc mẫu VLIW được hiển thị có khả năng phát hành đồng thời ba thao tác. Khe đầu tiên (nhóm bốn byte) dành cho branch instructions, khe giữa dành cho ALU instructions và khe cuối cùng dành cho load/store unit. Do ba hoạt động RISC cần thiết để thực hiện đoạn mã phụ thuộc (tính tuần tự), nên không thể đóng gói tải và thêm vào trong cùng một lệnh VLIW. Vì vậy, ba hướng dẫn VLIW riêng biệt là cần thiết.**

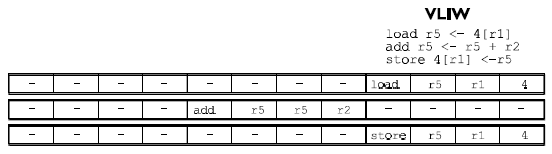
**With the code fragment as shown, the VLIW instruction is depressingly inefficient from the point of view of code destiny. In a real program situation, the compiler for the VLIW would use several program optimization techniques to fill all three slots in all three instructions. It is instructive to contemplate the performance each machine might achieve for this code. We need to assume that each machine has an efficient, pipelined implementation.**

**Với đoạn mã như được hiển thị, lệnh VLIW không hiệu quả theo quan điểm của code destiny. Trong tình huống chương trình thực, trình biên dịch cho VLIW sẽ sử dụng một số kỹ thuật tối ưu hóa chương trình để điền vào cả ba slots trong cả ba instructions. Hướng dẫn xem xét hiệu suất mà mỗi máy có thể đạt được cho mã này. Chúng ta cần giả định rằng mỗi máy có một sự triển khai hiệu quả, pipelined.**









**A CISC machine such as the 486 or Pentium would be able to execute the code fragment in three cycles.**

**Một máy CISC như 486 hoặc Pentium sẽ có thể thực thi đoạn mã theo ba chu kỳ.**

**A RISC machine would be able to execute the fragment in three cycles as well, but the cycles would likely be faster than on the CISC.**

**Một máy RISC cũng có thể thực hiện phân đoạn theo ba chu kỳ, nhưng các chu kỳ có thể sẽ nhanh hơn trên CISC.**

**The VLIW machine, assuming three fully-packed instructions, would effectively execute the code for this fragment in one cycle. To see this, observe that the fragment requires three out of nine slots, for one-third use of resources. One-third of three cycles is one cycle.**

**Máy VLIW, giả sử ba instructions được đóng gói đầy đủ, sẽ thực thi mã hiệu quả cho đoạn này trong một chu kỳ. Để thấy điều này, hãy quan sát rằng đoạn đó yêu cầu ba trong số chín vị trí, để sử dụng một phần ba tài nguyên. Một phần ba của ba chu kỳ là một chu kỳ.**

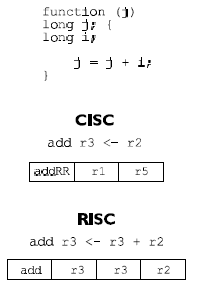
**--**

**To be even more accurate, we can assume good register allocation as shown in Figure 2. This example may actually be giving the CISC machine a slight unfair advantage since it will not be possible to allocate parameters to registers on the CISC as often as is possible for the RISC and VLIW.**

**Để chính xác hơn nữa, chúng ta có thể giả sử phân bổ thanh ghi tốt như trong Hình 2. Ví dụ này thực sự có thể mang lại cho máy CISC một lợi thế không công bằng một chút vì sẽ không thể phân bổ các tham số cho các thanh ghi trên CISC thường xuyên nhất có thể cho RISC và VLIW.**

**The CISC and RISC machines with good register allocation would take one cycle for one register-to-register instruction, but notice that the RISC code size is now much more in line with that of the CISC. Again assuming fully packed instructions, the VLIW execution time would also gain a factor of three benefit from good register allocation, yielding an effective execution time for the fragment of one-third of a cycle!**

**Các máy CISC và RISC có phân bổ register tốt sẽ mất một chu kỳ cho một lệnh đăng ký để đăng ký, nhưng lưu ý rằng kích thước mã RISC bây giờ phù hợp hơn nhiều so với CISC. Một lần nữa giả sử các hướng dẫn được đóng gói đầy đủ, thời gian thực hiện VLIW cũng sẽ đạt được hệ số ba lợi ích từ phân bổ đăng ký tốt, mang lại thời gian thực hiện hiệu quả cho một phần ba của chu kỳ!**

****

****

**Note that these comparisons have been between scalar (one-instruction per cycle maximum) RISC and CISC implementations and a relatively narrow VLIW. While it would be more realistic to compare superscalar RISCs and CISCs against a wider VLIW, such a comparison is more complicated. Suffice it to say that the conclusions would be roughly the same.**

**Lưu ý rằng các so sánh này là giữa các triển khai RISC và CISC vô hướng (một hướng dẫn trên mỗi chu kỳ) và một VLIW tương đối hẹp. Mặc dù sẽ thực tế hơn khi so sánh các RISC và CISC siêu thay thế so với VLIW rộng hơn, việc so sánh như vậy phức tạp hơn. Đủ để nói rằng các kết luận sẽ gần giống nhau.**

**SOFTWARE INSTEAD OF HARDWARE: IMPLEMENTATION ADVANTAGES OF VLIW**

**A VLIW implementation achieves the same effect as a superscalar RISC or CISC implementation, but the VLIW design does so without the two most complex parts of a high-performance superscalar design.**

**Việc triển khai VLIW đạt được hiệu quả tương tự như triển khai RISC hoặc CISC siêu lớp, nhưng thiết kế VLIW làm như vậy mà không có hai phần phức tạp nhất của thiết kế siêu lớp hiệu suất cao.**

**Because VLIW instructions explicitly specify several independent operations—that is, they explicitly, specify parallelism—it is not necessary to have decoding and dispatching hardware that tries to reconstruct parallelism from a serial instruction stream. Instead of having hardware attempt to discover parallelism, VLIW processors rely on the compiler that generates the VLIW code to explicitly specify parallelism. Relying on the compiler has advantages.**

**Bởi vì các lệnh của VLIW chỉ định rõ ràng một số hoạt động độc lập, nghĩa là, chúng chỉ định rõ rang song song, không cần thiết phải giải mã và gửi phần cứng cố gắng tái cấu trúc song song từ luồng lệnh nối tiếp. Thay vì có phần cứng cố gắng khám phá sự song song, bộ xử lý VLIW dựa vào trình biên dịch tạo mã VLIW để chỉ định rõ ràng sự song song. Dựa vào trình biên dịch có lợi thế.**

***https://whatis.techtarget.com/definition/VLIW-very-long-instruction-word***

**Very long instruction word (VLIW) describes a computer processing architecture in which a language compiler or pre-processor breaks program instruction down into basic operations that can be performed by the processor in parallel (that is, at the same time). These operations are put into a very long instruction word which the processor can then take apart without further analysis, handing each operation to an appropriate functional unit.**

**Từ lệnh rất dài (VLIW) mô tả kiến ​​trúc xử lý máy tính trong đó trình biên dịch ngôn ngữ hoặc bộ xử lý trước phá vỡ hướng dẫn chương trình thành các hoạt động cơ bản có thể được bộ xử lý thực hiện song song (nghĩa là đồng thời). Các hoạt động này được đưa vào một từ hướng dẫn rất dài mà sau đó bộ xử lý có thể tách rời mà không cần phân tích thêm, bàn giao từng thao tác cho một đơn vị chức năng thích hợp.**

**VLIW is sometimes viewed as the next step beyond the reduced instruction set computing ( RISC ) architecture, which also works with a limited set of relatively basic instructions and can usually execute more than one instruction at a time (a characteristic referred to as superscalar ). The main advantage of VLIW processors is that complexity is moved from the hardware to the software, which means that the hardware can be smaller, cheaper, and require less power to operate. The challenge is to design a compiler or pre-processor that is intelligent enough to decide how to build the very long instruction words. If dynamic pre-processing is done as the program is run, performance may be a concern.**

**VLIW đôi khi được xem là bước tiếp theo ngoài kiến ​​trúc điện toán tập lệnh giảm (RISC), cũng hoạt động với một tập lệnh giới hạn tương đối cơ bản và thường có thể thực thi nhiều hơn một lệnh tại một thời điểm (một đặc tính được gọi là siêu khối). Ưu điểm chính của bộ xử lý VLIW là độ phức tạp được chuyển từ phần cứng sang phần mềm, điều đó có nghĩa là phần cứng có thể nhỏ hơn, rẻ hơn và cần ít năng lượng hơn để vận hành. Thách thức là thiết kế một trình biên dịch hoặc bộ xử lý trước đủ thông minh để quyết định cách xây dựng các từ chỉ dẫn rất dài. Nếu tiền xử lý động được thực hiện khi chương trình được chạy, hiệu suất có thể là một mối quan tâm.**